Monotone Inductive and Coinductive Constructors of Rank 2

Ralph Matthes

Institut für Informatik der Ludwig-Maximilians-Universität München
Oettingenstraße 67, D-80538 München, Germany
matthes@informatik.uni-muenchen.de

March 22, 2001

Abstract. A generalization of positive inductive and coinductive types to monotone inductive and coinductive constructors of rank 1 and rank 2 is described. The motivation is taken from initial algebras and final coalgebras in a functor category and the Curry-Howard-correspondence. The definition of the system as a λ-calculus requires an appropriate definition of monotonicity to overcome subtle problems, most notably to ensure that the (co-)inductive constructors introduced via monotonicity of the underlying constructor of rank 2 are also monotone as constructors of rank 1. The problem is solved, strong normalization shown, and the notion proven to be wide enough to cover even highly pathological examples.

1 Introduction

Only in the last couple of years, the functional programming community has become interested in heterogeneous datatypes, i. e., polymorphic datatypes whose constructors relate different instances of the datatype—unlike the lists where cons takes a list of some type and produces a list of the same type. One might wonder whether there are interesting examples. It is by now well-known that de Bruijn notation indeed may be represented as a nested datatype [4, 2]. However, the question arose whether there are also examples taken from "the outside world". In [12], a nested datatype is shown which represents arbitrary square matrices over some type with elements accessible in logarithmic time. It uses type constructors of rank 2. On the theoretical side, [1] studies even coinductive type constructors of rank 2 with considerable nesting.

The present paper intends to give a thoroughly justified general framework for the description of terminating algorithms involving inductive and coinductive constructors of rank 2. It tries to be as general as possible while not departing from higher-order parametric polymorphism which—as a logical system—is higher-order intuitionistic propositional logic, expressed in natural deduction style. This (so-called Curry-Howard-)correspondence allows to insert a logical understanding into the system design: Inductive constructors $\mu F\mathcal{F}$ and $\nu F\mathcal{F}$ are not restricted to some kind of positivity, i. e., to requirements that free occurrences of F in $\mathcal{F}$ are only allowed an even number to the left of $\to$. Positivity is replaced by proven monotonicity: a term inhabiting the type expressing that $\lambda F \mathcal{F}$ is monotone. For rank 1 (inductive types) this has been studied at length in [10]—with the obvious notion of monotonicity expressible in system F. The main technical contribution is the definition of rank 2 monotonicity which will be justified at many places in this article.

The next section shortly reviews system F^ω, the long section 3 proposes and discusses the extension of F^ω to the system MICC^2 of monotonone inductive and coinductive constructors of rank 1 and rank 2. One of the most crucial properties is the subject of section 4: $\mu F \mathcal{F}$ and $\nu F \mathcal{F}$ are monotone if $\lambda F \mathcal{F}$ is. This result is not needed, however, for the reduction-preserving embedding into F^ω, shown in section 5. MICC^2's strong normalization follows from that of F^ω. Section 6 gives the idea of an extension even to primitive recursion of rank 2 while many closure properties of monotone constructors are shown in section 7. A few examples are considered in section 8.

2 System F^ω

Girard's system of higher-order parametric polymorphism [7] is an extension of system F where the set of types is extended to a simply-"typed" λ-calculus of constructors. The "types" of the constructors are called kinds and are built from the base kind $*$ (the universe of types) and the binary $\Rightarrow$ intended to form function kinds. We denote kinds by κ (possibly with indices). The constructors all have a kind and are built from kinded constructor variables as follows (constructor variables are denoted by $X, Y, Z, \ldots$ and constructors by $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \ldots$, constructors of kind $*$ are called types, and constructor variables of kind $*$ are (consequently) called type variables. They are denoted by $\alpha, \beta, \gamma, \ldots$ and types by $\rho, \sigma, \tau, \ldots$ The phrase "$\mathcal{X}$ is a constructor of kind κ" will be shortened to $\mathcal{X} : \kappa$, later also with superscript, i. e., $\mathcal{X}^\kappa$):

- $X^\kappa : \kappa$.
- If $\mathcal{X} : \kappa_2$ then $\lambda X^{\kappa_1} \mathcal{X} : \kappa_1 \Rightarrow \kappa_2$.
- If $\mathcal{X} : \kappa_1 \Rightarrow \kappa_2$ and $\mathcal{Y} : \kappa_1$ then $\mathcal{X}\mathcal{Y} : \kappa_2$.
- $\rho \to \sigma : *$, $\rho \times \sigma : *$, $\rho + \sigma : *$, $\forall X^\kappa \rho : *$.

Let $=_\beta$ be the (decidable) congruence closure of $(\lambda X^{\kappa_1} \mathcal{X}^{\kappa_2}) \mathcal{Y}^{\kappa_1} =_\beta \mathcal{X}[X := \mathcal{Y}]$. We identify constructors which are equal w. r. t. $=_\beta$.

Terms are built from typed variables and have a type (term variables are denoted x, y, z, u, v, f and terms $r, s, t, \ldots$ The phrase "r is a term of type ρ will be shortened to $r : \rho$, later also with superscript, i. e., r^ρ):

- $x^\rho : \rho$.
- If $r : \rho$ then $\Lambda X^\kappa r : \forall X^\kappa \rho$, provided X does not occur free in the types of the free term variables of r.
- If $r : \sigma$ then $\lambda x^\rho r : \rho \to \sigma$.

– If $r : \rho$ and $s : \sigma$ then $\langle r, s\rangle : \rho \times \sigma$.
– If $r : \forall X^\kappa \rho$ then $r\mathcal{X}^\kappa : \rho[X := \mathcal{X}]$.
– If $r : \rho \to \sigma$ and $s : \rho$ then $rs : \sigma$.
– If $r : \rho \times \sigma$ then $r\mathsf{L} : \rho$ and $r\mathsf{R} : \sigma$.

Finally, β-reduction $\rhd$ on terms is defined as the term closure of the following rules: $(\Lambda X^\kappa r)\mathcal{X} \rhd r[X := \mathcal{X}]$, $(\lambda x^\rho r)s \rhd r[x := s]$, $\langle r, s\rangle\mathsf{L} \rhd r$ and $\langle r, s\rangle\mathsf{R} \rhd s$.

Without being specific, we also assume to have term rules and β-reduction rules for the sum types $\rho + \sigma$ (they will not be used in the sequel).

$\Rightarrow$ for kinds and $\to$ for types shall associate to the right, while constructor application and term application are considered to be parenthesized to the left.

The rank of a kind is defined as expected:

$$\begin{aligned} \mathrm{rk}(*) &:= 0 \\ \mathrm{rk}(\kappa_1 \Rightarrow \kappa_2) &:= \max(\mathrm{rk}(\kappa_1) + 1, \mathrm{rk}(\kappa_2)) \end{aligned}$$

The rank of a constructor is defined to be the rank of its kind.

A useful example of a constructor of higher kind is the existential quantifier:

$$\exists_\kappa := \lambda X^{\kappa\Rightarrow *}\forall\alpha.(\forall Y^\kappa.XY \to \alpha) \to \alpha : (\kappa \Rightarrow *) \Rightarrow *.$$

The intuitionistic existential elimination is thus built into the definition, and the existential introduction is provided by the closed term

$$\Lambda X^{\kappa\Rightarrow *}\Lambda Y^\kappa \lambda x^{XY}\Lambda\alpha\lambda y^{\forall Y^\kappa.XY\to\alpha}.yYx : \forall X^{\kappa\Rightarrow *}\forall Y^\kappa.XY \to \exists_\kappa X.$$

Set $\exists X^\kappa \rho := \exists_\kappa(\lambda X \rho)$.

3 Definition of System MICC^2

System F^ω is extended by monotone inductive and coinductive constructors of rank 1 and rank 2, hence the name MICC^2.

Constructor variables of kind $* \Rightarrow *$ are denoted by $F, G, H, \ldots$ and constructors of kind $* \Rightarrow *$ by $\mathcal{F}, \mathcal{G}, \mathcal{H}, \ldots$ Define the abbreviation $\mathcal{F} \le \mathcal{G} := \forall\alpha.\mathcal{F}\alpha \to \mathcal{G}\alpha$.

We will only need monotonicity for the "pure" kinds $* \Rightarrow *$ and $(* \Rightarrow *) \Rightarrow * \Rightarrow *$ of rank 1 and 2, respectively.

$$\mathcal{F}^{*\Rightarrow *}\,\mathrm{mon} := \forall\alpha\forall\beta.(\alpha \to \beta) \to \mathcal{F}\alpha \to \mathcal{F}\beta$$

$$\boxed{\begin{aligned} &\mathcal{X}^{(*\Rightarrow *)\Rightarrow *\Rightarrow *}\,\mathrm{mon} := \Big(\forall F.F\,\mathrm{mon} \to \mathcal{X}F\,\mathrm{mon}\Big) \times \\ &\qquad \Big(\forall F\forall G.F \le G \to (F\,\mathrm{mon} \to \mathcal{X}F \le \mathcal{X}G) \times (G\,\mathrm{mon} \to \mathcal{X}F \le \mathcal{X}G)\Big) \end{aligned}}$$

There is no doubt that for rank 1 this is the expected and only reasonable definition. In contrast to that, there is a lot of freedom in giving a monotonicity definition for rank 2. Our definition is designed to meet the following criteria:

1. If F occurs only "positively" in some $\mathcal{F}$ then $\lambda F\mathcal{F}$ mon should be inhabited by one and only one "canonical" term.
2. Iteration and coiteration should be driven by the same monotonicity witnesses.
3. If $\lambda F\mathcal{F}$ mon is inhabited then also $(\mu F\mathcal{F})^{*\Rightarrow *}$ mon and $(\nu F\mathcal{F})^{*\Rightarrow *}$ mon should be inhabited in a "canonical" way.

However, the definition is slightly blown up in comparison with the following—intuitively isomorphic—possibility:

$$\Big(\forall F.F\,\mathrm{mon} \to \mathcal{X}F\,\mathrm{mon}\Big) \times \Big(\forall F\forall G.F \leq G \to (F\,\mathrm{mon} + G\,\mathrm{mon}) \to \mathcal{X}F \leq \mathcal{X}G\Big)$$

Both definitions are intuitively isomorphic since, in general, $\rho_1 + \rho_2 \to \sigma$ and $(\rho_1 \to \sigma) \times (\rho_2 \to \sigma)$ are. However, in applications we would need to refer to full extensional equality for sum types (even beyond permutative conversions) which is still decidable [6] but difficult to treat since it leaves the realm of rewriting. Since our focus is on intensional equality as expressed by term rewriting, the slightly less elegant definition had to be made.

First, we extend system F^ω by monotone inductive constructors of rank 1, i. e., inductive types $\mu\alpha\rho$ depending on the monotonicity of the constructor $\lambda\alpha\rho$ of rank 1. This is exactly done as for the iterative part of the elimination-based system EMIT of monotone inductive types in [10]: For every type ρ we assume the type $\mu\alpha\rho$ in the system of constructors (and that $\mu\alpha$ binds every free occurrence of α in ρ).[1] The type $\mu\alpha\rho$ is called a monotone inductive constructor of rank 1 (or monotone inductive type) although its *formation* does not at all rest on a monotonicity assumption. This notion is justified since it is the usage of those constructors as specified in the *term* formation rules which constitutes a constructor concept. In our case, this is made clear in the introduction rule (corresponding to fold in the functional programming literature) since we introduce the type $\mu\alpha\rho$ by

$$\frac{m : \lambda\alpha\rho\,\mathrm{mon} \qquad t : \rho[\alpha := \mu\alpha\rho]}{\mathsf{C}_{\mu\alpha\rho} m t : \mu\alpha\rho}\,(\mu\text{--}I)$$

(This rule is—following the standard practice in type theory—to be read as: If m is a term of type $\lambda\alpha\rho\,\mathrm{mon}$ and t is a term of type $\rho[\alpha := \mu\alpha\rho]$ then $\mathsf{C}_{\mu\alpha\rho} m t$ is a term of type $\mu\alpha\rho$. Thus, $\mathsf{C}_{\mu\alpha\rho}$ is considered as a binary function symbol.)

If there is a closed $m : \lambda\alpha\rho\,\mathrm{mon}$, we call $\mu\alpha\rho$ a *proven* monotone inductive constructor of rank 1 (or proven monotone inductive type) and m a monotonicity *witness* of $\lambda\alpha\rho$. The type $\mu\alpha\rho$ is eliminated by an iterative concept:

$$\frac{r : \mu\alpha\rho \qquad s : \rho[\alpha := \sigma] \to \sigma}{r\mathsf{E}_\mu s : \sigma}\,(\mu\text{--}E)$$

For $s : \rho[\alpha := \sigma] \to \sigma$ define $\mathcal{E}_\mu s := \lambda x^{\mu\alpha\rho}.x\mathsf{E}_\mu s : \mu\alpha\rho \to \sigma$.[2]

[1] Alternatively, one could have assumed a constructor constant $\mu : (* \Rightarrow *) \Rightarrow *$ of rank 2 and set $\mu\alpha\rho := \mu(\lambda\alpha\rho)$.

[2] Why don't we introduce $\mathcal{E}_\mu s$ as official syntax for the μ-elimination rule? Because we favour natural deduction style as opposed to Hilbert style. For studies on fixed-points in λ-calculi this seems to be a reasonable choice, see also [15].

The β-rule of *iteration* is

$$(\mathsf{C}_{\mu\alpha\rho}mt)\mathsf{E}_\mu s \rhd s\Big(m(\mu\alpha\rho)\sigma(\mathcal{E}_\mu s)t\Big)$$

These rules obviously generalize the system of positive inductive types, e. g., system $2\lambda\mathsf{J}$ of Leivant [9] in which $\mu\alpha\rho$ is only allowed if α only occurs "positively" in ρ. In our language, all those $\mu\alpha\rho$ are proven monotone inductive types.[3] Therefore, the iteration rule for $2\lambda\mathsf{J}$ simply makes use of those predetermined monotonicity witnesses. In [10, 4.2.2 and 5.1.1], the embedding of (a variant of) $2\lambda\mathsf{J}$ into the present system is carried out in all details (even with primitive recursion instead of iteration only).

A more categorical motivation (still with fixed monotonicity witnesses) may be found in [5]: $\mu\alpha\rho$ may be conceived as a weakly initial $\lambda\alpha\rho$-algebra (for rank 2, this is carried out below).

Now we also extend the system by monotone inductive constructors of rank 2, i. e., inductive constructors $\mu F\mathcal{F}$ depending on the monotonicity of the constructor $\lambda F\mathcal{F}$ of rank 2. The constructor $\mu F\mathcal{F}$ has kind $* \Rightarrow *$. Hence, as a constructor, $\mu F\mathcal{F}$ is of rank 1, but it is called a monotone inductive constructor of rank 2. More precisely, we extend the system of constructors by another constructor formation rule: If $\mathcal{F}$ is a constructor of kind $* \Rightarrow *$ then $\mu F\mathcal{F}$ is a constructor of kind $* \Rightarrow *$, and free occurrences of F in $\mathcal{F}$ are bound by μF.[4] The constructor $\mu F\mathcal{F}$ is called a monotone inductive constructor of rank 2 although again its formation does not at all rest on a monotonicity assumption. Since $\mu F\mathcal{F}$ is not a type, the introduction and eliminination rules describe how to introduce and how to eliminate the types $(\mu F\mathcal{F})\rho$ for arbitrary ρ. We start with the introduction rule:

$$\frac{m : \lambda F\mathcal{F}\,\mathrm{mon} \qquad t : \mathcal{F}[F := \mu F\mathcal{F}]\rho}{\mathsf{C}_{\mu F\mathcal{F}}mt : (\mu F\mathcal{F})\rho}{\scriptstyle(\mu-I)}$$

For $m : \lambda F\mathcal{F}\,\mathrm{mon}$ define

$$\mathcal{C}_{\mu F\mathcal{F}}m := \Lambda\alpha\lambda x^{\mathcal{F}[F:=\mu F\mathcal{F}]\alpha}.\mathsf{C}_{\mu F\mathcal{F}}mx : \mathcal{F}[F := \mu F\mathcal{F}] \leq \mu F\mathcal{F}.$$

If there is a closed $m : \lambda F\mathcal{F}\,\mathrm{mon}$, we call $\mu F\mathcal{F}$ a *proven* monotone inductive constructor of rank 2 and m a monotonicity *witness* of $\lambda F\mathcal{F}$.[5]

The elimination rule (corresponding to an iterator) is:

$$\frac{r : (\mu F\mathcal{F})\rho \qquad n : \mathcal{G}\,\mathrm{mon} \qquad s : \mathcal{F}[F := \mathcal{G}] \leq \mathcal{G}}{r\mathsf{E}_\mu ns : \mathcal{G}\rho}{\scriptstyle(\mu-E)}$$

---

[3] This is an easy consequence of theorem 3 below.

[4] Analogous to rank 1, we could have assumed a constant $\mu : ((* \Rightarrow *) \Rightarrow * \Rightarrow *) \Rightarrow * \Rightarrow *$ of rank 3 and set $\mu F\mathcal{F} := \mu(\lambda F\mathcal{F})$.

[5] In Theorem 1 below it is proved that if $\mu F\mathcal{F}$ is a proven monotone constructor of rank 2, then $\mu F\mathcal{F}$—as a constructor of rank 1—is monotone (of rank 1), and hence $\mu\alpha.(\mu F\mathcal{F})\alpha$ becomes a proven monotone inductive constructor of rank 1 which, as a constructor, is simply a type, hence of rank 0.

For $n : \mathcal{G}\,\mathrm{mon}$ and $s : \mathcal{F}[F := \mathcal{G}] \leq \mathcal{G}$ define

$$\mathcal{E}_\mu ns := \Lambda\alpha\lambda x^{(\mu F\mathcal{F})\alpha}.x\mathsf{E}_\mu ns : \mu F\mathcal{F} \leq \mathcal{G}.$$

The β-rule of *iteration* for rank 2 is

$$(\mathsf{C}_{\mu F\mathcal{F}}mt)\mathsf{E}_\mu ns \rhd s\rho\Big(m\mathsf{R}(\mu F\mathcal{F})\mathcal{G}(\mathcal{E}_\mu ns)\mathsf{R}n\rho t\Big)$$

Some comments are in order. First we collect the information contained in the term rules while ignoring the terms themselves. We get the following inference rules:

$$\frac{\lambda F\mathcal{F}\,\mathrm{mon}}{\mathcal{F}[F := \mu F\mathcal{F}] \leq \mu F\mathcal{F}}(\mu\!-\!I) \qquad \text{and} \qquad \frac{\mathcal{G}\,\mathrm{mon} \qquad \mathcal{F}[F := \mathcal{G}] \leq \mathcal{G}}{\mu F\mathcal{F} \leq \mathcal{G}}(\mu\!-\!E)$$

This means that $\mu F\mathcal{F}$ is a pre-fixed-point of $\lambda F\mathcal{F}$ (provided $\lambda F\mathcal{F}$ is monotone) and that it is less than or equal to any monotone pre-fixed-point $\mathcal{G}$ of $\lambda F\mathcal{F}$ (whose monotonicity is not required this time). In case $\lambda F\mathcal{F}$ is monotone, we will prove below (see footnote 5) that $\mu F\mathcal{F}$ is monotone, too, whence $\mu F\mathcal{F}$ becomes a minimal monotone pre-fixed-point of $\lambda F\mathcal{F}$. Thus, $\mu F\mathcal{F}$ follows the lattice-theoretic understanding of inductive definitions. But there are also the terms and the term rewrite rule of iteration. Viewing category theory as coherently constructive lattice theory [3], we can justify the iteration rule by reference to the concept of a weakly initial algebra as has been done for inductive *types* in [5] mentioned earlier. We will do this only rather sloppily. The "category" consists of the monotone constructors of kind $* \Rightarrow *$, the "morphisms" from $\mathcal{F}$ to $\mathcal{G}$ being the terms of type $\mathcal{F} \leq \mathcal{G}$, and the "composition" being the functional composition of terms. An "endofunctor" is a monotone $\lambda F\mathcal{F}$, in this situation a "$\lambda F\mathcal{F}$-algebra" is a monotone $\mathcal{G}$ together with a term $s : \mathcal{F}[F := \mathcal{G}] \leq \mathcal{G}$. The constructor $\mu F\mathcal{F}$ now turns out to be a *weakly*[6] initial $\lambda F\mathcal{F}$-algebra, i. e., $\mu F\mathcal{F}$ together with $\mathcal{C}_{\mu F\mathcal{F}}m$ is a $\lambda F\mathcal{F}$-algebra—as expressed by the introduction rule (concerning monotonicity of $\mu F\mathcal{F}$, see footnote 5 once more). And, whenever $\mathcal{G}$ and s form a $\lambda F\mathcal{F}$-algebra, which amounts to monotonicity of $\mathcal{G}$—witnessed by a term $n : \mathcal{G}\,\mathrm{mon}$—and $s : \mathcal{F}[F := \mathcal{G}] \leq \mathcal{G}$, then there is the term $\mathcal{E}_\mu ns$ which is a morphism from $\mu F\mathcal{F}$ to $\mathcal{G}$, such that the diagram in figure 1 "commutes". In that diagram the expression $\mathcal{F}[F := \mathcal{E}_\mu ns]$ shall denote the action of the functor $\lambda F\mathcal{F}$ on the morphism $\mathcal{E}_\mu ns$. The most natural understanding is given by the following definition: $\mathcal{F}[F := \mathcal{E}_\mu ns] := m\mathsf{R}(\mu F\mathcal{F})\mathcal{G}(\mathcal{E}_\mu ns)\mathsf{R}n$.[7] Therefore, commutation of the diagram means that for any type ρ and any term $t : \mathcal{F}[F := \mu F\mathcal{F}]\rho$, we have

$$\mathcal{E}_\mu ns\rho(\mathcal{C}_{\mu F\mathcal{F}}m\rho t) = s\rho\Big(m\mathsf{R}(\mu F\mathcal{F})\mathcal{G}(\mathcal{E}_\mu ns)\mathsf{R}n\rho t\Big).$$

Obviously, the left-hand side is β-equivalent (using only rules already present in system F) with the left-hand side of the iteration rule, and the right-hand side

[6] Full initiality includes a uniqueness statement and would lead to an extensional notion of equality.

[7] Notice how our definition of monotonicity for rank 2 allows to avoid the reference to the monotonicity of $\mu F\mathcal{F}$.

$$\begin{tikzcd}
\mathcal{F}[F := \mu F\mathcal{F}] \arrow[r, "\mathcal{C}_{\mu F\mathcal{F}} m"] \arrow[d, "\mathcal{F}[F:=\mathcal{E}_\mu ns]"'] & \mu F\mathcal{F} \arrow[d, "\mathcal{E}_\mu ns"] \\
\mathcal{F}[F := \mathcal{G}] \arrow[r, "s"'] & \mathcal{G}
\end{tikzcd}$$

Fig. 1. Weakly initial $\lambda F\mathcal{F}$-algebra

is that of the iteration rule. Subject reduction will now also be clear, i. e., the right-hand side of the iteration rule receives the same type as the left-hand side.

By dualization, we also introduce monotone coinductive constructors of rank 1 and rank 2: Let also $\nu\alpha\rho$ be a type and $\nu F\mathcal{F}$ be a constructor of kind $* \Rightarrow *$ (with binding as for $\mu\alpha\rho$ and $\mu F\mathcal{F}$). The type $\nu\alpha\rho$ is called a monotone coinductive constructor of rank 1 (a monotone coinductive type), and $\nu F\mathcal{F}$ is called a monotone coinductive constructor of rank 2. The term formation rules pertaining to $\nu\alpha\rho$ are an elimination rule (corresponding to unfold in the functional programming literature) and a mechanism for coiteration. We start with the elimination rule:

$$\frac{r : \nu\alpha\rho \qquad m : \lambda\alpha\rho\,\mathrm{mon}}{r\mathsf{E}_\nu m : \rho[\alpha := \nu\alpha\rho]}(\nu{-}E)$$

If there is a monotonicity witness of $\lambda\alpha\rho$, we call $\nu\alpha\rho$ a *proven* monotone coinductive constructor of rank 1 (or proven monotone coinductive type). Terms of type $\nu\alpha\rho$ are introduced by a coiterative concept:

$$\frac{s : \sigma \to \rho[\alpha := \sigma] \qquad t : \sigma}{\mathsf{C}_{\nu\alpha\rho} st : \nu\alpha\rho}(\nu{-}I)$$

For $s : \sigma \to \rho[\alpha := \sigma]$ define $\mathcal{C}_{\nu\alpha\rho} s := \lambda x^\sigma.\mathsf{C}_{\nu\alpha\rho} sx : \sigma \to \nu\alpha\rho$.

The β-rule of *coiteration* is

$$(\mathsf{C}_{\nu\alpha\rho} st)\mathsf{E}_\nu m \rhd m\sigma(\nu\alpha\rho)(\mathcal{C}_{\nu\alpha\rho} s)(st)$$

Rank 2 is treated analogously. The elimination rule is

$$\frac{r : (\nu F\mathcal{F})\rho \qquad m : \lambda F\mathcal{F}\,\mathrm{mon}}{r\mathsf{E}_\nu m : \mathcal{F}[F := \nu F\mathcal{F}]\rho}(\nu{-}E)$$

For $m : \lambda F\mathcal{F}\,\mathrm{mon}$ define

$$\mathcal{E}_\nu m := \Lambda\alpha\lambda x^{(\nu F\mathcal{F})\alpha}.x\mathsf{E}_\nu m : \nu F\mathcal{F} \leq \mathcal{F}[F := \nu F\mathcal{F}].$$

If there is a monotonicity witness of $\lambda F\mathcal{F}$, we call $\nu F\mathcal{F}$ a *proven* monotone coinductive constructor of rank 2.

The types $(\nu F\mathcal{F})\rho$ are introduced via:

$$\frac{n : \mathcal{G}\,\mathrm{mon} \qquad s : \mathcal{G} \leq \mathcal{F}[F := \mathcal{G}] \qquad t : \mathcal{G}\rho}{\mathsf{C}_{\nu F\mathcal{F}} nst : (\nu F\mathcal{F})\rho}(\nu{-}I)$$

For $n : \mathcal{G}\,\mathrm{mon}$ and $s : \mathcal{G} \le \mathcal{F}[F := \mathcal{G}]$ define

$$\mathcal{C}_{\nu F \mathcal{F}} ns := \Lambda\alpha\lambda x^{\mathcal{G}\alpha}.\mathsf{C}_{\nu F \mathcal{F}} nsx : \mathcal{G} \le \nu F\mathcal{F}.$$

Finally, the β-rule of *coiteration* for rank 2 becomes:

$$(\mathsf{C}_{\nu F\mathcal{F}} nst)\mathsf{E}_\nu m \rhd m\mathsf{R}\mathcal{G}(\nu F\mathcal{F})(\mathcal{C}_{\nu F\mathcal{F}} ns)\mathsf{L}n\rho(s\rho t)$$

It is again worth studying these notions in the light of category theory. Firstly, the system without terms has the following rules:

$$\frac{\lambda F\mathcal{F}\,\mathrm{mon}}{\nu F\mathcal{F} \le \mathcal{F}[F := \nu F\mathcal{F}]}(\nu{-}E) \qquad \text{and} \qquad \frac{\mathcal{G}\,\mathrm{mon} \qquad \mathcal{G} \le \mathcal{F}[F := \mathcal{G}]}{\mathcal{G} \le \nu F\mathcal{F}}(\nu{-}I)$$

Hence, $\nu F\mathcal{F}$ is a post-fixed-point of $\lambda F\mathcal{F}$ if $\lambda F\mathcal{F}$ is monotone and that it is greater or equal to any monotone post-fixed-point of $\lambda F\mathcal{F}$ (even without the assumption of $\lambda F\mathcal{F}$'s monotonicity). Since $\nu F\mathcal{F}$ is also monotone (see theorem 1), $\nu F\mathcal{F}$ is a maximal monotone post-fixed-point of $\lambda F\mathcal{F}$ provided $\lambda F\mathcal{F}$ is monotone. Therefore, it is appropriate to call $\nu F\mathcal{F}$ a coinductive constructor. Concerning terms and term rewrite rules, if we again have an endofunctor, i.e., a monotone $\lambda F\mathcal{F}$, a "$\lambda F\mathcal{F}$-coalgebra" is a monotone constructor $\mathcal{G}$ (which constructively means that we have a term $n : \mathcal{G}\,\mathrm{mon}$) together with a term $s : \mathcal{G} \le \mathcal{F}[F := \mathcal{G}]$. The constructor $\nu F\mathcal{F}$ is a weakly final $\lambda F\mathcal{F}$-coalgebra: It is a $\lambda F\mathcal{F}$-coalgebra and for any $\lambda F\mathcal{F}$-coalgebra, given by $\mathcal{G}$, $n : \mathcal{G}\,\mathrm{mon}$ and s, the term $\mathcal{C}_{\nu F\mathcal{F}} ns$ is a morphism from $\mathcal{G}$ to $\nu F\mathcal{F}$ such that the diagram in figure 2 commutes. This time, the easiest definition of the action of the functor on such a

$$\begin{tikzcd}
\mathcal{F}[F := \nu F\mathcal{F}] & \nu F\mathcal{F} \arrow[l, "\mathcal{E}_\nu m"'] \\
\mathcal{F}[F := \mathcal{G}] \arrow[u, "{\mathcal{F}[F:=\mathcal{C}_{\nu F\mathcal{F}} ns]}"] & \mathcal{G} \arrow[l, "s"] \arrow[u, "\mathcal{C}_{\nu F\mathcal{F}} ns"']
\end{tikzcd}$$

Fig. 2. Weakly final $\lambda F\mathcal{F}$-coalgebra

morphism is given by $\mathcal{F}[F := \mathcal{C}_{\nu F\mathcal{F}} ns] := m\mathsf{R}\mathcal{G}(\nu F\mathcal{F})(\mathcal{C}_{\nu F\mathcal{F}} ns)\mathsf{L}n$ since it avoids using $\nu F\mathcal{F}$'s monotonicity. Hence, the possibility to choose whether to use the monotonicity of the first or second argument to $m\mathsf{R}$ is vital to fulfill criterion 2 in the list on page 4.

Observe that the comparison of the figures 1 and 2 displays the duality between $\mu F\mathcal{F}$ and $\nu F\mathcal{F}$ much better than that of the rules of iteration and coiteration.

This completes the definition of the extension of system F^ω by monotone inductive and coinductive constructors of rank 1 and rank 2, henceforth called system MICC^2. (Although the different extensions have been described separately, it is understood that they are meant to be done simultaneously which implies, e. g., the presence of the type $\mu\alpha.(\nu F\mathcal{F})\alpha$ in our system for arbitrary constructors $\mathcal{F}$.)

4 Monotone (Co-)Inductive Constructors are Monotone

Monotone inductive and monotone coinductive constructors of rank 2 are constructors $\mu F\mathcal{F}$ and $\nu F\mathcal{F}$ having rank 1. Therefore, we may ask whether they are in turn monotone (of rank 1) if $\lambda F\mathcal{F}$ were monotone (of rank 2). This is answered in the affirmative, and consequently, e. g., $\nu\alpha.(\mu F\mathcal{F})\alpha$ is a proven monotone coinductive type if $\mu F\mathcal{F}$ is a proven monotone inductive constructor.

Theorem 1. *There are closed terms* $M_{\mu F\mathcal{F}} : \lambda F\mathcal{F}\,\mathrm{mon} \to \mu F\mathcal{F}\,\mathrm{mon}$ *and* $M_{\nu F\mathcal{F}} : \lambda F\mathcal{F}\,\mathrm{mon} \to \nu F\mathcal{F}\,\mathrm{mon}$.

This shows that criterion 3 in the list on page 4 is met (the construction is "canonical" since the monotonicity witness is not analyzed at all—it is merely fed in as an argument).

Proof. We argue slightly informally by assuming a term $m : \lambda F\mathcal{F}\,\mathrm{mon}$ and showing $\mu F\mathcal{F}\,\mathrm{mon}$ and $\nu F\mathcal{F}\,\mathrm{mon}$ using m. We begin with the inductive constructor. Set

$$\mathcal{G} := \lambda\alpha\forall\beta.(\alpha \to \beta) \to (\mu F\mathcal{F})\beta.$$

In order to prove monotonicity of $\mu F\mathcal{F}$, it suffices to show that $\mu F\mathcal{F} \leq \mathcal{G}$. Of course, this is done by proving that $\mathcal{G}$ is a $\lambda F\mathcal{F}$-algebra. We have the canonical monotonicity witness

$$n := \Lambda\alpha\Lambda\gamma\lambda f^{\alpha\to\gamma}\lambda x^{\mathcal{G}\alpha}\Lambda\beta\lambda y^{\gamma\to\beta}.x\beta(\lambda v^{\alpha}.y(fv)) : \mathcal{G}\,\mathrm{mon}$$

(α occurs only non-strictly positively in $\forall\beta.(\alpha \to \beta) \to (\mu F\mathcal{F})\beta$). Set

$$\ell := \Lambda\alpha\lambda x^{\mathcal{G}\alpha}.x\alpha(\lambda y^{\alpha}y) : \mathcal{G} \leq \mu F\mathcal{F} \qquad \text{and}$$

$$s := \Lambda\alpha\lambda x^{\mathcal{F}[F:=\mathcal{G}]\alpha}\Lambda\beta\lambda f^{\alpha\to\beta}.\mathsf{C}_{\mu F\mathcal{F}}m(m\mathsf{R}\mathcal{G}(\mu F\mathcal{F})\ell\mathsf{L}n\beta(m\mathsf{L}\mathcal{G}n\alpha\beta fx))$$

which has type $\mathcal{F}[F := \mathcal{G}] \leq \mathcal{G}$. The idea is: $\lambda F\mathcal{F}$ is monotone, hence also $\mathcal{F}[F := \mathcal{G}]$. Therefore, $\alpha \to \beta$ implies $\mathcal{F}[F := \mathcal{G}]\alpha \to \mathcal{F}[F := \mathcal{G}]\beta$. Thus, $m\mathsf{L}\mathcal{G}n\alpha\beta fx : \mathcal{F}[F := \mathcal{G}]\beta$. Since $\lambda F\mathcal{F}$ is monotone and $\mathcal{G} \leq \mu F\mathcal{F}$, we have $\mathcal{F}[F := \mathcal{G}] \leq \mathcal{F}[F := \mu F\mathcal{F}]$, which is the type of $m\mathsf{R}\mathcal{G}(\mu F\mathcal{F})\ell\mathsf{L}n$. Consequently, the second argument to $\mathsf{C}_{\mu F\mathcal{F}}$ gets type $\mathcal{F}[F := \mu F\mathcal{F}]\beta$, hence the term starting with $\mathsf{C}_{\mu F\mathcal{F}}$ has type $(\mu F\mathcal{F})\beta$. The proof is completed by

$$\Lambda\alpha\Lambda\beta\lambda f^{\alpha\to\beta}\lambda x^{(\mu F\mathcal{F})\alpha}.x\mathsf{E}_{\mu}ns\beta f : \mu F\mathcal{F}\,\mathrm{mon}$$

since $x\mathsf{E}_{\mu}ns$ has type $\mathcal{G}\alpha$.

We turn to the coinductive constructor. $\mathcal{G}$ is the greatest monotone constructor which is less than or equal to $\mu F\mathcal{F}$. Dually, we seek for the least monotone constructor which is greater than or equal to $\nu F\mathcal{F}$.[8] Instead of taking $\lambda\beta\exists\alpha.(\alpha\to\beta)\times(\nu F\mathcal{F})\alpha$, we avoid the product type and set

$$\mathcal{H} := \lambda\beta\forall\gamma.\Big(\forall\alpha.(\alpha\to\beta)\to(\nu F\mathcal{F})\alpha\to\gamma\Big)\to\gamma.$$

We will recycle the names n, ℓ, s. Since $\mathcal{H}$ is also non-strictly positive, it has a canonical mononicity witness

$$n := \Lambda\beta\Lambda\delta\lambda f^{\beta\to\delta}\lambda x^{\mathcal{H}\beta}\Lambda\gamma\lambda y^{\forall\alpha.(\alpha\to\delta)\to(\nu F\mathcal{F})\alpha\to\gamma}.x\gamma(\Lambda\alpha\lambda u^{\alpha\to\beta}.y\alpha(\lambda v^{\alpha}.f(uv)))$$

Set $\ell := \Lambda\beta\lambda x^{(\nu F\mathcal{F})\beta}\Lambda\gamma\lambda z^{\forall\alpha.(\alpha\to\beta)\to(\nu F\mathcal{F})\alpha\to\gamma}.z\beta(\lambda y^{\beta}y)x : \nu F\mathcal{F}\leq\mathcal{H}$ and show that we have got a $\lambda F\mathcal{F}$-coalgebra by finding a term of type $\mathcal{H}\leq\mathcal{F}[F:=\mathcal{H}]$:

$$s := \Lambda\beta\lambda x^{\mathcal{H}\beta}.x(\mathcal{F}[F:=\mathcal{H}]\beta)(\Lambda\alpha\lambda f^{\alpha\to\beta}\lambda z^{(\nu F\mathcal{F})\alpha}t)$$

where we used $t := m\mathsf{L}\mathcal{H}n\alpha\beta f\Big(m\mathsf{R}(\nu F\mathcal{F})\mathcal{H}\ell\mathsf{R}n\alpha(z\mathsf{E}_{\nu}m)\Big) : \mathcal{F}[F:=\mathcal{H}]\beta$. (Note that $z\mathsf{E}_{\nu}m$ has type $\mathcal{F}[F:=\nu F\mathcal{F}]\alpha$ and that the idea for finding t is essentially the same as in the proof of $\mu F\mathcal{F}$'s monotonicity.) The proof is finished with

$$\Lambda\alpha\Lambda\beta\lambda f^{\alpha\to\beta}\lambda x^{(\nu F\mathcal{F})\alpha}.\mathsf{C}_{\nu F\mathcal{F}}ns(\Lambda\gamma\lambda y^{\forall\alpha.(\alpha\to\beta)\to(\nu F\mathcal{F})\alpha\to\gamma}.y\alpha f x) : \nu F\mathcal{F}\,\mathrm{mon}$$

which is again a bit weird due to the impredicative encoding of the existential quantifier. $\square$

Notice that for the proofs it has been essential to be free to decide whether to provide monotonicity for the first or the second constructor argument when using monotonicity of $\lambda F\mathcal{F}$. In the proof, the other argument has always been that for which we tried to prove monotonicity! Obviously, one cannot require both arguments to be monotone and get the proof through. Neither can we dispense with monotonicity of the arguments altogether since we would not get, e. g., a witness for $\lambda F\lambda\alpha.F(F\alpha)\,\mathrm{mon}$ and hence arrive at too narrow a concept. If we required monotonicity only of the first argument, then the rule of iteration would have to refer to the monotonicity of $\mu F\mathcal{F}$. Although our monotonicity proof would still go through, this considerably complex term would enormously blow up the terms during iteration (not to speak about the deviation from the wanted behaviour). On the contrary, if we required monotonicity only of the second argument, iteration would not have to be changed, but there would be no way to recover the proof of monotonicity for $\mu F\mathcal{F}$. Moreover, the situation for coiteration would always be the other way round, and it would hardly be

[8] The lattice-theoretic version of $\mathcal{G}$ is called the lower monotonization of an operator in [10, p.83]. Its dual (the "upper monotonization" of an operator), when written in type theory, gives $\mathcal{H}$. Both play a central role in understanding inductive and coinductive types à la Mendler, see [11] for the definition, [15] for the explanation and [10] for applications to monotone inductive types.

acceptable if different notions of monotonicity were required for the treatment of inductive and coinductive constructors.

The categorial perspective did not at all suggest our definition of monotonicity for rank 2. Categorially, one would consider the category of endofunctors (see [1] for a recent treatment of rank 2 in this framework) and would never speak about other entities whereas we also make requirements for non-monotone arguments. And, of course, monotonicity is only a part of functoriality, since "coherence" is not dealt with: functor laws cannot be expressed in our type system.

5 MICC^2 Embeds into System F^ω

It is well-known that iteration and coiteration may be encoded impredicatively, and that this can be done such that strong normalization of the system with iteration and coiteration can be inferred from that of the impredicative system, see e. g. [5, 15]. This also worked easily for monotone inductive types [13, 10]. Hence, it does not come as a surprise that MICC^2 embeds into F^ω.

Theorem 2. *There is an embedding $-'$ of MICC^2 into F^ω, i. e., for every constructor $\mathcal{X}$ of kind κ in MICC^2, there is a constructor $\mathcal{X}'$ of kind κ in F^ω, and for every term r of type ρ in MICC^2, there is a term r' of type ρ' in F^ω such that whenever $r \rhd s$ in MICC^2, then $r' \rhd^+ s'$ in F^ω (every step in MICC^2 is translated into at least one step in F^ω).*

Proof. The definition is by iteration on constructors and by iteration on terms, respectively. We only consider the most interesting non-homomorphic clauses. Note that only (co-)inductive constructors are affected and that we therefore have $(\mathcal{F} \leq \mathcal{G})' = \mathcal{F}' \leq \mathcal{G}'$ and $(\mathcal{X}\,\mathrm{mon})' = \mathcal{X}'\,\mathrm{mon}$. It will also be clear that $(\mathcal{X}[X := \mathcal{Y}])' = \mathcal{X}'[X := \mathcal{Y}']$, $(r[X := \mathcal{Y}])' = r'[X := \mathcal{Y}']$ and $(r[x^\rho := s])' = r'[x^{\rho'} := s']$. Define

$$\begin{aligned}
(\mu\alpha\rho)' &:= \forall\alpha.(\rho \to \alpha) \to \alpha \\
(r\mathsf{E}_\mu s)' &:= r'\sigma' s' \\
(\mathsf{C}_{\mu\alpha\rho} m t)' &:= \Lambda\alpha\lambda x^{\rho\to\alpha}.x\Big(m'(\mu\alpha\rho)'\alpha(\lambda z^{(\mu\alpha\rho)'}.z\alpha x)t'\Big) \\
(\mu F\mathcal{F})' &:= \lambda\alpha\forall F.F\,\mathrm{mon} \to \mathcal{F}' \leq F \to F\alpha \\
(r\mathsf{E}_\mu n s)' &:= r'\mathcal{G}'n's' \\
(\mathsf{C}_{\mu F\mathcal{F}} m t)' &:= \Lambda F\lambda y^{F\,\mathrm{mon}}\lambda x^{\mathcal{F}'\leq F}.x\rho'\Big(m'\mathsf{R}(\mu F\mathcal{F})'F(\Lambda\alpha\lambda z^{(\mu F\mathcal{F})'\alpha}.zFyx)\mathsf{R}y\rho't'\Big)
\end{aligned}$$

We do not set $(\nu\alpha\rho)'$ to $\exists\alpha.(\alpha \to \rho') \times \alpha$ but avoid the product type by the following definitions:

$$\begin{aligned}
(\nu\alpha\rho)' &:= \forall\beta.(\forall\alpha.(\alpha \to \rho') \to \alpha \to \beta) \to \beta \\
(\mathsf{C}_{\nu\alpha\rho} s t)' &:= \Lambda\beta\lambda z^{\forall\alpha.(\alpha\to\rho')\to\alpha\to\beta}.z\sigma' s' t' \\
(r\mathsf{E}_\nu m)' &:= r'(\rho[\alpha := \nu\alpha\rho])'\Big(\Lambda\alpha\lambda z_1^{\alpha\to\rho'}\lambda z_2^\alpha.m'\alpha(\nu\alpha\rho)' \\
&\qquad (\lambda x^\alpha\Lambda\beta\lambda z^{\forall\alpha.(\alpha\to\rho')\to\alpha\to\beta}.z\alpha z_1 x)(z_1 z_2)\Big)
\end{aligned}$$

$$
\begin{aligned}
(\nu F\mathcal{F})' &:= \lambda\alpha\forall\beta.(\forall F.F\,\mathrm{mon} \to F \leq \mathcal{F}' \to F\alpha \to \beta) \to \beta \\
(\mathsf{C}_{\nu F\mathcal{F}}nst)' &:= \Lambda\beta\lambda z^{\forall F.F\,\mathrm{mon}\to F\leq\mathcal{F}'\to F\rho'\to\beta}.z\mathcal{G}'n's't' \\
(r\mathsf{E}_\nu m)' &:= r'(\mathcal{F}[F := \nu F\mathcal{F}]\rho)'(\Lambda F\lambda y^{F\,\mathrm{mon}}\lambda z_1^{F\leq\mathcal{F}'}\lambda z_2^{F\rho'}.m'\mathsf{R}F(\nu F\mathcal{F})' \\
&\quad \Big(\Lambda\alpha\lambda x^{F\alpha}\Lambda\beta\lambda z^{\forall F.F\,\mathrm{mon}\to F\leq\mathcal{F}'\to F\alpha\to\beta}.zFyz_1x)\mathsf{L}y\rho'(z_1\rho'z_2)\Big)
\end{aligned}
$$

It is not too hard to check the translation of steps of MICC^2 into those of F^ω. □

Corollary 1. *System* MICC^2 *is strongly normalizing, i. e.,* $\rhd$ *is well-founded.*

Proof. It is well-known that F^ω is strongly normalizing ([7] only considered weak normalization although his proof method copes with strong normalization as well) and, certainly, strong normalization is inherited via embeddings. □

In this impredicative encoding, the previous section's concern becomes irrelevant since $(\mu F\mathcal{F})'$ and $(\nu F\mathcal{F})'$ are always monotone regardless of monotonicity of $\lambda F\mathcal{F}$. In some sense, α occurs only strictly positively in the kernel of $(\mu F\mathcal{F})'$ and only non-strictly positively in the kernel of $(\nu F\mathcal{F})'$. In fact, we have

$$\Lambda\alpha\Lambda\beta\lambda f^{\alpha\to\beta}\lambda x^{(\mu F\mathcal{F})'\alpha}\Lambda F\lambda y^{F\,\mathrm{mon}}\lambda z^{\mathcal{F}'\leq F}.y\alpha\beta f(xFyz) : (\mu F\mathcal{F})'\,\mathrm{mon}$$

$$\Lambda\alpha\Lambda\gamma\lambda f^{\alpha\to\gamma}\lambda x^{(\nu F\mathcal{F})'\alpha}\Lambda\beta\lambda u^{\forall F.F\,\mathrm{mon}\to F\leq\mathcal{F}'\to F\gamma\to\beta}.x\beta t : (\nu F\mathcal{F})'\,\mathrm{mon}$$

with

$$t := \Lambda F\lambda y^{F\,\mathrm{mon}}\lambda z^{F\leq\mathcal{F}'}\lambda v^{F\alpha}.uFyz(y\alpha\gamma fv) : \forall F.F\,\mathrm{mon} \to F \leq \mathcal{F}' \to F\alpha \to \beta.$$

In the general lattice-theoretic situation (that of Tarski's fixed-point theorem), this phenomenon also occurs: The pointwise infimum of all the pre-fixed-points of an operator on a lattice of monotone set-theoretic functions is automatically monotone.

6 Extension by Primitive (Co-)Recursion

System MICC^2 may now easily be extended by the concept of primitive recursion which is extremely unlikely to have a reasonable embedding into system F^ω [14]. We just have to add another elimination rule for $\mu F\mathcal{F}$ and a new β-reduction rule of primitive recursion. We abbreviate $\mathcal{F}\times\mathcal{G} := \lambda\alpha.\mathcal{F}\alpha\times\mathcal{G}\alpha$.

$$\frac{r : (\mu F\mathcal{F})\rho \qquad s : \mathcal{F}[F := \mu F\mathcal{F}\times\mathcal{G}] \leq \mathcal{G}}{r\mathsf{E}_\mu^+ s : \mathcal{G}\rho}\,(\mu\text{-}E^+)$$

For $s : \mathcal{F}[F := \mu F\mathcal{F}\times\mathcal{G}] \leq \mathcal{G}$ define

$$\mathcal{E}_\mu^+ s := \Lambda\alpha\lambda x^{(\mu F\mathcal{F})\alpha}.x\mathsf{E}_\mu^+ s : \mu F\mathcal{F} \leq \mathcal{G}.$$

The β-rule of *primitive recursion* for rank 2 is

$$(\mathsf{C}_{\mu F\mathcal{F}}mt)\mathsf{E}_\mu^+ s \rhd s\rho\Big(m\mathsf{R}(\mu F\mathcal{F})(\mu F\mathcal{F}\times\mathcal{G})(\Lambda\alpha\lambda x^{(\mu F\mathcal{F})\alpha}.\langle x,(\mathcal{E}_\mu^+ s)\alpha x\rangle)\mathsf{L}(M_{\mu F\mathcal{F}}m)\rho t\Big)$$

Notice that we no longer require $\mathcal{G}\,\mathrm{mon}$ since we anyway have to refer to $\mu F\mathcal{F}$'s monotonicity as expressed by $M_{\mu F\mathcal{F}}$.

The respective treatment for rank 1 in [10] is now obvious:

$$\frac{r : \mu\alpha\rho \qquad s : \rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma}{r\mathsf{E}_\mu^+ s : \sigma}(\mu{-}E^+)$$

For $s : \rho[\alpha := \sigma] \to \sigma$ define $\mathcal{E}_\mu^+ s := \lambda x^{\mu\alpha\rho}.x\mathsf{E}_\mu^+ s : \mu\alpha\rho \to \sigma$. The β-rule of primitive recursion of rank 1 is

$$(\mathsf{C}_{\mu\alpha\rho} mt)\mathsf{E}_\mu^+ s \rhd s\Big(m(\mu\alpha\rho)(\mu\alpha\rho \times \sigma)(\lambda x^{\mu\alpha\rho}.\langle x, (\mathcal{E}_\mu^+ s)x\rangle)t\Big)$$

In [10] it is shown that this schema may be embedded into that of primitive recursion only for non-strictly positive inductive types. Also, this primitive recursion captures the intuitive notion of primitive recursion. In the rank 2 case, the author has no knowlegde of interesting examples. Therefore, also the dualization to corecursion is left as an exercise.

7 Proven Monotone (Co-)Inductive Constructors

There is a whealth of constructors having monotonicity witnesses. Below a list of closure properties will be given. It turns out, however, that this wealth cannot be grasped without the companion notion of anti-monotonicity. Somewhat surprising, we only need it for rank 1.

$$\mathcal{F}\,\mathrm{mon}^- := \forall\alpha\forall\beta.(\alpha \to \beta) \to \mathcal{F}\beta \to \mathcal{F}\alpha.$$

Aiming at a concise description, let p always range over the set of *polarities* $\{+, -\}$, set $-- := +$ and $-+ := -$ and $\mathcal{F}\,\mathrm{mon}^+ := \mathcal{F}\,\mathrm{mon}$. Hence $\mathcal{F}\,\mathrm{mon}^{\mathrm{p}}$ and $\mathcal{F}\,\mathrm{mon}^{-\mathrm{p}}$ are always types.

Theorem 3. *There are closed terms of the following types:*

1. $\lambda\alpha\rho\,\mathrm{mon}^{\mathrm{p}}$ *if* α *does not occur free in* ρ
2. $\lambda\alpha\alpha\,\mathrm{mon}^+$
3. $\lambda\alpha\rho\,\mathrm{mon}^{\mathrm{p}} \to \lambda\alpha\sigma\,\mathrm{mon}^{\mathrm{p}} \to \lambda\alpha.\rho \times \sigma\,\mathrm{mon}^{\mathrm{p}}$
4. $\lambda\alpha\rho\,\mathrm{mon}^{\mathrm{p}} \to \lambda\alpha\sigma\,\mathrm{mon}^{\mathrm{p}} \to \lambda\alpha.\rho + \sigma\,\mathrm{mon}^{\mathrm{p}}$
5. $\lambda\alpha\rho\,\mathrm{mon}^{-\mathrm{p}} \to \lambda\alpha\sigma\,\mathrm{mon}^{\mathrm{p}} \to \lambda\alpha.\rho \to \sigma\,\mathrm{mon}^{\mathrm{p}}$
6. $(\forall\gamma.\lambda\alpha\rho\,\mathrm{mon}^{\mathrm{p}}) \to \lambda\alpha\forall\gamma\rho\,\mathrm{mon}^{\mathrm{p}}$
7. $(\forall\alpha.\lambda\gamma\rho\,\mathrm{mon}^+) \to (\forall\gamma.\lambda\alpha\rho\,\mathrm{mon}^{\mathrm{p}}) \to \lambda\alpha\mu\gamma\rho\,\mathrm{mon}^{\mathrm{p}}$
8. $(\forall\alpha.\lambda\gamma\rho\,\mathrm{mon}^+) \to (\forall\gamma.\lambda\alpha\rho\,\mathrm{mon}^{\mathrm{p}}) \to \lambda\alpha\nu\gamma\rho\,\mathrm{mon}^{\mathrm{p}}$
9. $\lambda F\lambda\alpha\rho\,\mathrm{mon} \to \lambda\alpha\rho\,\mathrm{mon}^+$ *if* F *does not occur free in* $\lambda\alpha\rho$
10. $\lambda F\mathcal{F}\,\mathrm{mon} \to (\forall F.F\,\mathrm{mon} \to \forall\gamma(\lambda\alpha.\mathcal{F}\gamma\,\mathrm{mon})) \to \lambda\alpha.(\mu F\mathcal{F})\alpha\,\mathrm{mon}^+$
11. $\lambda F\mathcal{F}\,\mathrm{mon} \to (\forall F.F\,\mathrm{mon} \to \forall\gamma(\lambda\alpha.\mathcal{F}\gamma\,\mathrm{mon})) \to \lambda\alpha.(\nu F\mathcal{F})\alpha\,\mathrm{mon}^+$
12. $\lambda\alpha\rho\,\mathrm{mon}^+ \to \lambda F\lambda\alpha\rho\,\mathrm{mon}$ *if* F *does not occur free in* $\lambda\alpha\rho$
13. $\lambda FF\,\mathrm{mon}$
14. $\lambda F\mathcal{F}\,\mathrm{mon} \to \lambda F\mathcal{G}\,\mathrm{mon} \to \lambda F.\mathcal{F} \times \mathcal{G}\,\mathrm{mon}$

15. $\lambda F\mathcal{F}\,\mathrm{mon} \to \lambda F\mathcal{G}\,\mathrm{mon} \to \lambda F.\mathcal{F} + \mathcal{G}\,\mathrm{mon}$ *with* $\mathcal{F} + \mathcal{G} := \lambda\alpha.\mathcal{F}\alpha + \mathcal{G}\alpha$
16. $(\forall F.\lambda G\mathcal{F}\,\mathrm{mon}) \to (\forall G.\lambda F\mathcal{F}\,\mathrm{mon}) \to \lambda F\mu G\mathcal{F}\,\mathrm{mon}$
17. $(\forall F.\lambda G\mathcal{F}\,\mathrm{mon}) \to (\forall G.\lambda F\mathcal{F}\,\mathrm{mon}) \to \lambda F\nu G\mathcal{F}\,\mathrm{mon}$
18. $\lambda F\mathcal{F}\,\mathrm{mon} \to \lambda F\mathcal{G}\,\mathrm{mon} \to \lambda F\lambda\alpha.\mathcal{F}(\mathcal{G}\alpha)\,\mathrm{mon}$

An informal claim is the "canonicity" of all those proofs as long as their goal does not degenerate to be covered by 1 or 12.[9]

Proof. The first 8 properties are quite common since they form the basis of the definition of canonical monotonicity witnesses for positive inductive and coinductive types (see e. g. [9, 8, 5, 10]).

Rules 10 and 11 are proved by first establishing $\forall\gamma(\lambda\alpha.(\mu F\mathcal{F})\gamma\,\mathrm{mon})$ and $\forall\gamma(\lambda\alpha.(\nu F\mathcal{F})\gamma\,\mathrm{mon})$, using the assumptions. For this to work, one has to make use of theorem 1.

The other rules are more or less adaptations of those for rank 1 to rank 2.

The interesting new case is 18. We assume $m_1 : \lambda F\mathcal{F}\,\mathrm{mon}$, $m_2 : \lambda F\mathcal{G}\,\mathrm{mon}$ and show $\lambda F\lambda\alpha.\mathcal{F}(\mathcal{G}\alpha)\,\mathrm{mon}$. Clearly, the proof will be a pair. Its first component having type $\forall F.F\,\mathrm{mon} \to \lambda\alpha.\mathcal{F}(\mathcal{G}\alpha)\,\mathrm{mon}$ is

$$\Lambda F\lambda x^{F\,\mathrm{mon}}\lambda\alpha\lambda\beta\lambda f^{\alpha\to\beta}.m_1\mathsf{L}Fx(\mathcal{G}\alpha)(\mathcal{G}\beta)(m_2\mathsf{L}Fx\alpha\beta f)$$

We abbreviate $\mathcal{F}' := \mathcal{F}[F := G]$ and $\mathcal{G}' := \mathcal{G}[F := G]$. Then the second component is given by $\Lambda F\Lambda G\lambda x^{F\leq G}.\langle\lambda y^{F\,\mathrm{mon}}\Lambda\alpha.t_1, \lambda y^{G\,\mathrm{mon}}\Lambda\alpha.t_2\rangle$ with t_1 and t_2 of type $\mathcal{F}(\mathcal{G}\alpha) \to \mathcal{F}'(\mathcal{G}'\alpha)$, defined by

$$t_1 := \lambda z^{\mathcal{F}(\mathcal{G}\alpha)}.m_1\mathsf{R}FGx\mathsf{L}y(\mathcal{G}'\alpha)\Big(m_1\mathsf{L}Fy(\mathcal{G}\alpha)(\mathcal{G}'\alpha)(m_2\mathsf{R}FGx\mathsf{L}y\alpha)z\Big)$$

$$t_2 := \lambda z^{\mathcal{F}(\mathcal{G}\alpha)}.m_1\mathsf{L}Gy(\mathcal{G}\alpha)(\mathcal{G}'\alpha)(m_2\mathsf{R}FGx\mathsf{R}y\alpha)(m_1\mathsf{R}FGx\mathsf{R}y(\mathcal{G}\alpha)z)$$

Their ideas are as follows: In t_1 we first show $\mathcal{G}\alpha \to \mathcal{G}'\alpha$, then lift this to $\mathcal{F}(\mathcal{G}\alpha) \to \mathcal{F}(\mathcal{G}'\alpha)$. We compose this with $\mathcal{F}(\mathcal{G}'\alpha) \to \mathcal{F}'(\mathcal{G}'\alpha)$. In this case, $\mathcal{F}'\,\mathrm{mon}$ is not available. In t_2, we show $\mathcal{F}(\mathcal{G}\alpha) \to \mathcal{F}'(\mathcal{G}\alpha)$, then we lift $\mathcal{G}\alpha \to \mathcal{G}'\alpha$ to $\mathcal{F}'(\mathcal{G}\alpha) \to \mathcal{F}'(\mathcal{G}'\alpha)$ and finally compose. Here, $\mathcal{F}\,\mathrm{mon}$ is not available. This shows that although two different strategies for proving $\mathcal{F}(\mathcal{G}\alpha) \to \mathcal{F}'(\mathcal{G}'\alpha)$ exist, we have no choice when producing t_1 and t_2. $\square$

Unfortunately, through rule 18, we can get different monotonicity witnesses since repeated application can be decomposed in different ways: The witness for $\lambda F\lambda\alpha.\mathcal{F}(\mathcal{G}(\mathcal{H}\alpha))\,\mathrm{mon}$ may be derived from $\lambda F\mathcal{F}\,\mathrm{mon}$ and $\lambda F\lambda\alpha.\mathcal{G}(\mathcal{H}\alpha)\,\mathrm{mon}$ or from $\lambda F\lambda\alpha.\mathcal{F}(\mathcal{G}\alpha)$ and $\lambda F\mathcal{H}\,\mathrm{mon}$, and hence both from $\lambda F\mathcal{F}\,\mathrm{mon}$, $\lambda F\mathcal{G}\,\mathrm{mon}$ and $\lambda F\mathcal{H}\,\mathrm{mon}$. By careful inspection, it becomes clear that two different strategies are followed which are not equal unless the canonical witness for $\mathcal{F}\,\mathrm{mon}$ (inside the proof) fulfills the second functor law (compatibility with composition). In general, this cannot be expected for an intensional notion of equality. Hence, criterium 1 from the list on page 4 could not be fully satisfied.

[9] This could be formulated slightly better by referring to universally quantified types such as $\forall X^{(*\Rightarrow*)\Rightarrow*\Rightarrow*}\forall Y^{(*\Rightarrow*)\Rightarrow*\Rightarrow*}.X\,\mathrm{mon} \to Y\,\mathrm{mon} \to \lambda F\lambda\alpha.XF(YF\alpha)\,\mathrm{mon}$ instead of 18.

8 Examples, Conclusions and Future Work

We start with an example not to be found from theorem 3 alone: There is a witness for $\lambda F \lambda\alpha.(\lambda\alpha.(((\alpha \to \gamma) \to \alpha) \to \alpha) \to \alpha)(F(F\alpha))\,\mathrm{mon}$. This follows from theorem 3 as soon as we give a monotonicity witness for the constructor $\lambda\alpha.(((\alpha \to \gamma) \to \alpha) \to \alpha) \to \alpha$, proposed by U. Berger as the first example of a non-positive and monotone constructor of rank 1: The last but one occurrence of α is at a negative position. Nevertheless, it is not hard to find a monotonicity witness. Set $0 := \mu\alpha\alpha$ and $1 := \nu\alpha\alpha$ which are proven monotone.

Untyped λ-calculus may be represented with de Bruijn indices, and this can be reflected in the constructor $\mathcal{F} := \mu F \lambda\alpha.\alpha + F(1+\alpha) + (F\alpha \times F\alpha)$. $\mathcal{F}\rho$ would be the type of untyped λ-terms having free variables taken from ρ. Clearly, theorem 3 implies that $\mathcal{F}$ is a proven monotone inductive constructor. By iteration on $\mathcal{F}$, substitution can be defined. The result is the same as that of the structurally inductive approach in [2]. The representation is also studied in [4] but with the concept of generalized folds. [4] also exhibits a variation which in our notation reads $\mu F \lambda\alpha.\alpha + F(1 + F\alpha) + (F\alpha \times F\alpha)$. No doubt, theorem 3 shows that this is a proven monotone inductive constructor. Another proven monotone variation would be $\mu F \lambda\alpha.\alpha + F(1 + \alpha) + (F\alpha \times F\alpha) + ((F\alpha \to 0) \to 0)$ which would add a kind of continuation terms to the untyped λ-calculus.

As an example of the main result in [1], functional programs for the computation of the isomorphisms between $(\mu\alpha.1+\alpha\times\alpha) \to \sigma$ and $(\nu F \lambda\alpha.\alpha \times F(F\alpha))\sigma$ are given. Obviously, both $\mu\alpha.1+\alpha\times\alpha$ and $\nu F \lambda\alpha.\alpha \times F(F\alpha)$ are proven monotone by theorem 3. By iteration and coiteration, the "isomorphisms" can be established and have as reduction behaviour that described in program in [1] which is based on pattern-matching. While [1] establishes that those are in fact isomorphisms w. r. t. semantic equality we cannot expect such a result for our intensional setting. However, since MICC^2 is strongly normalizing, those functional programs are guaranteed to terminate.

Notice that these examples did not use the possibility of interleaved μ and ν. However, the author still does not know useful examples for rank 2 using interleaving.

Concluding remark: The proposed notion of monotonicity for rank 2 seems to be appropriate to base systems of inductive and coinductive constructors on it.

Open questions: We are still in need of a good notion of positivity for rank 2. Can we expect monotonicity witnesses for positive constructors to be "unique"? It would be interesting to study course-of-value iteration for rank 2 or the extension of other concepts in [15] to rank 2. The relation to [4]'s generalized folds should be studied carefully. Can their behaviour be simulated in MICC^2? Can one find data structures which need deeper nested inductive constructors than the example of square matrices in [12]? Is there a chance to get a similar clean view on constructors of higher rank than 2?

References

1. Thorsten Altenkirch. Representations of first order function types as terminal coalgebras. In Samson Abramsky, editor, *TLCA01*, Lecture Notes in Computer Science. Springer Verlag, 2001. To appear.
2. Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer Verlag, 1999.
3. Roland Backhouse and Marcel Bijsterveld. Category theory as coherently constructive lattice theory: An illustration. CS report 94-43, Eindhoven University of Technology, November 1994.
4. Richard S. Bird and Ross Paterson. De Bruijn notation as a nested datatype. *Journal of Functional Programming*, 9(1):77–91, 1999.
5. Herman Geuvers. Inductive and coinductive types with iteration and recursion. In Bengt Nordström, Kent Pettersson, and Gordon Plotkin, editors, *Proceedings of the 1992 Workshop on Types for Proofs and Programs, Båstad, Sweden, June 1992*, pages 193–217, 1992. Only published electronically. Available at `ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z`.
6. Neil Ghani. $\beta\eta$-equality for coproducts. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications (TLCA '95), Edinburgh, United Kingdom, April 1995*, volume 902 of *Lecture Notes in Computer Science*, pages 171–185. Springer Verlag, 1995.
7. Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* Thèse de Doctorat d'État, Université de Paris VII, 1972.
8. Brian Howard. *Fixed Points and Extensionality in Typed Functional Programming Languages.* PhD thesis, Stanford University, 1992.
9. Daniel Leivant. Contracting proofs to programs. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 279–327. Academic Press, 1990.
10. Ralph Matthes. *Extensions of System* F *by Iteration and Primitive Recursion on Monotone Inductive Types.* Doktorarbeit (PhD thesis), University of Munich, 1998.
11. Nax P. Mendler. Recursive types and type constraints in second-order lambda calculus. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science, Ithaca, N.Y.*, pages 30–36. IEEE Computer Society Press, 1987.
12. Chris Okasaki. From fast exponentiation to square matrices: An adventure in types. In *Proceedings of the fourth ACM SIGPLAN International Conference on Functional Programming (ICFP '99), Paris, France, September 27-29, 1999*, volume 34 of *SIGPLAN Notices*, pages 28–35. ACM, 1999.
13. Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur.* Habilitation à diriger les recherches, ENS Lyon, 1996.
14. Zdzisław Spławski and Paweł Urzyczyn. Type Fixpoints: Iteration vs. Recursion. In *Proceedings of the fourth ACM SIGPLAN International Conference on Functional Programming (ICFP '99), Paris, France, September 27-29, 1999*, volume 34 of *SIGPLAN Notices*, pages 102–113. ACM, 1999.
15. Tarmo Uustalu and Varmo Vene. Least and greatest fixed points in intuitionistic natural deduction. *Theoretical Computer Science*, 2000. To appear.